Using Models at Runtime For Monitoring and Adaptation of Networked Physical Devices: Example of a Flexible Manufacturing System

Mathieu Vallée[1], Munir Merdan[2], and Thomas Moser[3]

[1] Institute of Computer Technologies
[2] Automation and Control Institute
[3] Institute for Software Technology and Interactive Systems
Vienna University of Technology, Austria
{firstname.lastname}@tuwien.ac.at

Abstract. The emergence of software-intensive systems connecting physical devices to network-based applications involves new design challenges. As an example, flexible manufacturing systems composed of multiple networked devices in interaction with the physical world, are subject to imprecision and to unpredictable breakdowns. Applications and control software are therefore highly complex, and must operate in heterogeneous and rapidly changing environments.
To address these issues, we describe an approach using models at runtime for efficiently monitoring and adapting the software controlling mechatronic devices. We consider a decentralized system, in which each device is represented as an agent. Each agent maintains a model integrating a representation of itself, of its environment and of the agent society, and uses this model to detect inconsistencies, to envision possible future states and to create explanations based on past states. In this paper, we focus on presenting our model and highlighting the results, benefits and challenges arising from using models at run-time with networked physical devices.

1 Introduction

The emergence of software-intensive systems connecting physical devices to network-based applications offers new exciting possibilities. Among them, flexible manufacturing systems providing a faster, more efficient response to market changes are envisioned [10]. However, engineering such complex systems operating in heterogeneous and rapidly changing environments poses numerous challenges. Traditional control approaches cannot cope with new requirements, due to their rigidity and limited capability for agile adaptation to unexpected internal and external disturbances [8]. The application of decentralized control architectures, based on autonomous and co-operative agents, is considered as a promising approach. Intelligent agents offer a convenient way of modeling processes that are distributed over space and time, making the control of the system decentralized [7], increasing flexibility and enhances fault tolerance. Using agent-based

software for controlling a flexible manufacturing system has been largely investigated in the recent years. However, most work focus on planning and scheduling issues [8] (disconnected from the actual control of physical devices [3]) or use simple, reactive agents for producing specialized adaptation behaviors. Currently, concerns about robustness and stability prevent the wide industrial adoption of these initial solutions [15]. Advances on self-awareness, self-adaptation and self-healing are needed [4].

To address these issues, we proposed an approach in which agents use models for efficiently monitoring and adapting the software controlling mechatronic devices. Each agent manages a model integrating a representation of itself, of its environment and of the agent society, and enabling it to detect inconsistencies, to envision possible future states and to create explanations based on past states. In recent works [18, 11, 9], we showed how this approach supports complex tasks such as detection of anomalies of sensor reading, failure recovery and runtime reconfiguration. In this paper, we focus on presenting our use of models at runtime with networked physical devices and discussing the challenges arising from this approach.

This paper is structured as follows. Section 2 introduces some background and example about distributed intelligent control of a flexible manufacturing system. Section 3 details the world model of an automation agent, forming the central piece of our approach. Section 4 discusses our results and lessons learned. Section 5 discussed related work and section 6 concludes with a summary.

2 Background: Distributed Intelligent Control of a Flexible Pallet Transfer System

As an example for monitoring and adapting software in interaction with networked physical devices, our current studies focuses on flexible manufacturing systems. In this paper, we use the example of a pallet transport system, located in the Odo-Struder-Laboratory[4]. Due to their role to connect different parts of the system and to carry and route materials between them, transportation systems are in most cases seen as a key element, but the increasing need for more flexibility significantly complicates the control of these systems.

Overview of the Pallet Transfer System The pallet transfer system (Fig. 1) consists of software-controlled manufacturing components: transport components such as conveyor belts (dark green lines) and diverters (yellow circles); and assembly machines (colored rectangles with round corners). Product parts are transported on pallets (colored rectangles; colors represent the target machines). Each pallet carries an RFID tag providing information on its destinations, which diverters can access through RFID readers (rectangles on conveyors). Figure 2

[4] Industrial automation systems laboratory of the Automation and Control Institute, Vienna University of Technology

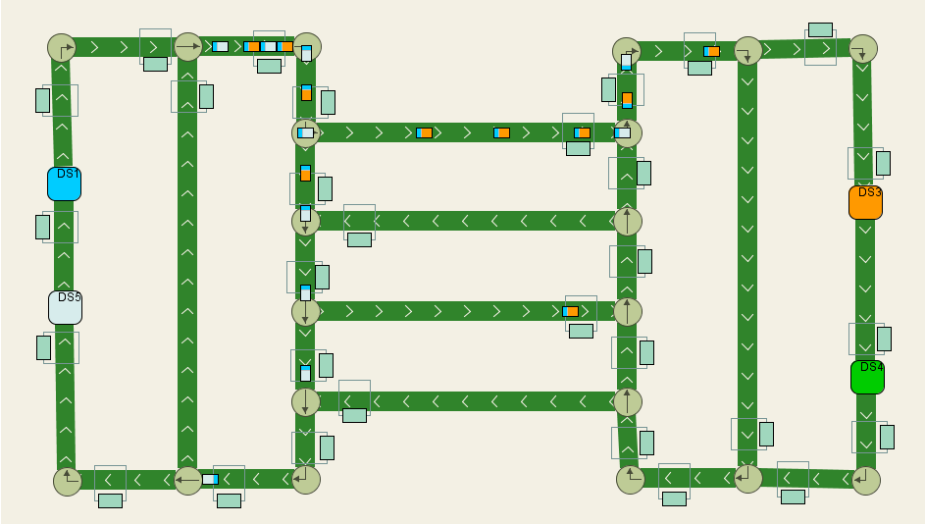
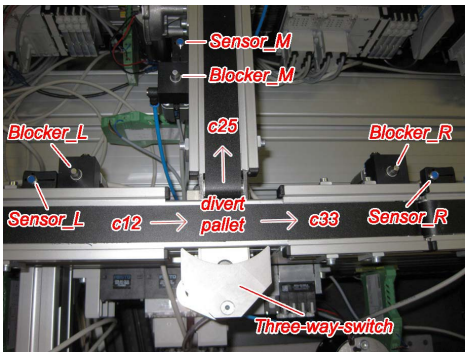
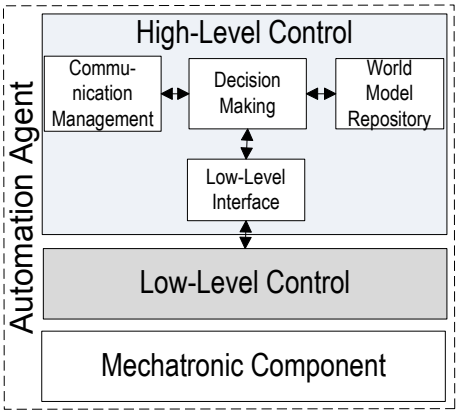


Fig. 1. Overview of the pallet transfer system



(a) Mechatronic component (diverter)



(b) Architecture of the automation agent

Fig. 2. Automation agent controlling a diverter

(a) depicts the mechatronic component realizing a diverter. It is mainly composed of a switch (directing a pallet), sensors (detecting the presence of a pallet) and blockers (preventing a pallet from moving).

Architecture of an Automation Agent In order to support the design of distributed intelligent control software for manufacturing systems, we introduced a generic architecture for automation agents in [17]. The architecture is depicted on Fig. 2 (b), and consists of two software layers, in addition to the mechatronic component. The low-level control (LLC) layer is in charge of controlling the hardware. The high-level control (HLC) layer is in charge of diagnostics, of coordination with other agents and of self-adaption based on the representation of the world. In the case of the diverter, a "diverter agent" contains a LLC layer responsible for moving the switch depending on the destination of on incoming pallets, and a HLC layer responsible for , e.g, redefining routes in response to disturbances at other components or validating sensor readings with information from other agents. Distinguishing LLC and HLC within each automation agent is fundamental for devices in interaction with the physical world. In our architecture, the LLC is responsible for performing all necessary operations in real-time, while the HLC is only responsible for non-functional monitoring and adaptation, which might require longer computation time and interactions with others agents or even with human operators. To enforce this layering while enabling efficient adaptation, we base our LLC on the IEC 61499 standard, enhanced with programmable reconfigurations capabilities [22, 9].

Figure 2 (b) also depicts more precisely the four main modules composing the inner architecture of the HLC. The world model repository contains a world model, i.e., a symbolic representation of the world of the agent. The low-level interface enables the HLC to monitor and to adapt the LLC. It especially provides facilities for receiving event notifications about the current operations of the LLC and for requesting reconfiguration in the LLC. The communication manager provides facilities for managing the communication with other agents. The decision-making component is in charge of coordinating the reasoning about states of the world and deciding what to do (e.g., communicate with other machines, request an operation from the LLC, issue notifications to an operator). Event notifications generated by the LLC, by communication with other agents or by the world model trigger the decision-making procedures.

3 World Model of an Automation Agent

The world model plays a central role in the architecture of an automation agent. In this section, we describe its content and illustrate it using the example of the diverter agent.

3.1 Properties

The world model has to provide two key properties. Firstly it should *integrate information about different views* of the world, which are sometimes overlapping.

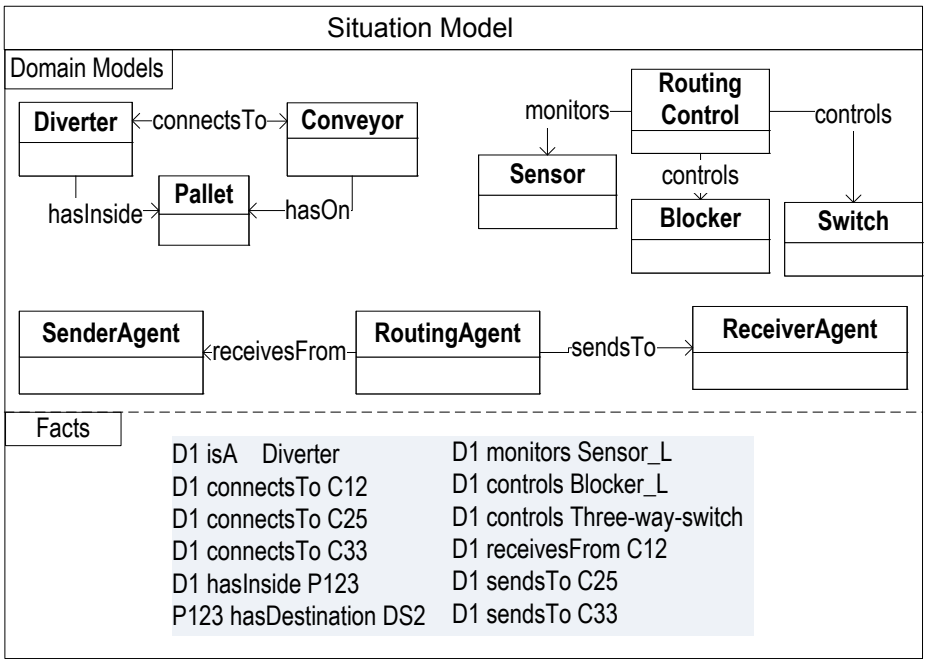


Fig. 3. Situation Model for the diverter agent

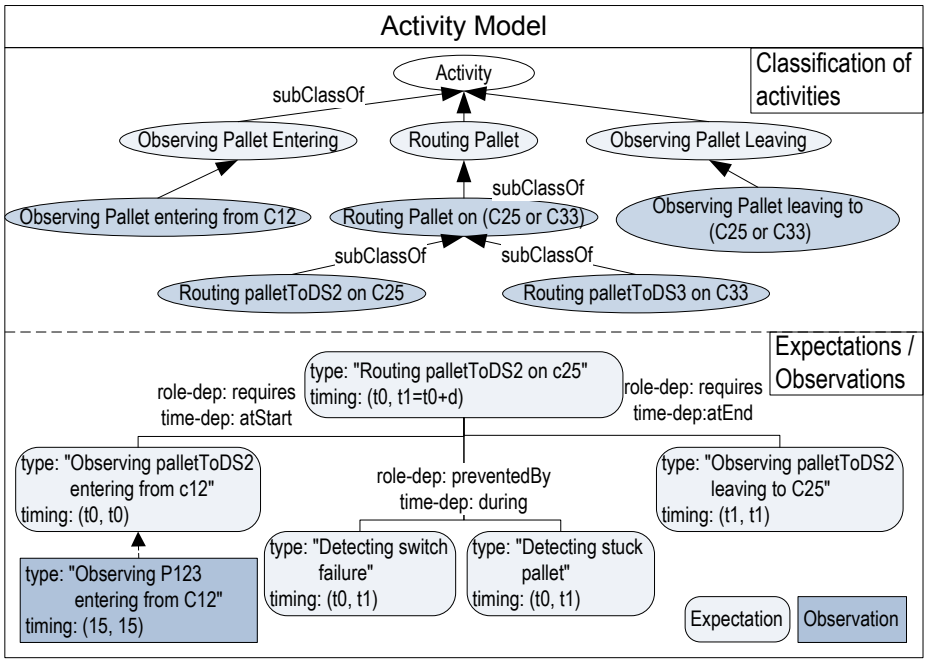


Fig. 4. Activity Model for the diverter agent

It must integrate information about the state of its environment (the physical world in which it is evolving), about the agent society (the other agents with which it is interacting, their roles and tasks), and about its own internal structure and processes. Clearly, this is related to reflection, so the representation of the agent itself is a key element of the world model [17].

Secondly and most importantly, the world model must support a *flexible synchronization* with the real world. In particular, it is in general not possible to assume that the model provides a complete and up-to-date view on the real world. We must cope with partial and scarce observations, and we use models to compensate for the lack of direct information with assumptions about what the state of the world should be. As a consequence, the model must be validated and revised anytime new information is received. More precisely, it should provide three key features:

- The *detection of inconsistencies* between the current world model and new information (received from LLC or other agents). In a real world setting, inconsistencies may arise both from inaccuracy of the model on the one side, and from imprecision of the information sources on the other side.
- The *derivation of possible future states* of the world and their relevant characteristics (answering "what-if" questions). It should be possible to define expectations about the future state of the world, and to plan meaningful observations accordingly. It should also be possible to envision multiple possibilities about the future in order to be prepared for adaptation.
- The *derivation of explanations* from past states of the world (answering "why" questions). Although all information about the world may not be accessible, models can be used to explain current observations with assumption about past states of the world which could not be observed directly. In some casesw, this can trigger additional observation to confirm assumptions. This is particularly useful for diagnosis, when root causes for failures can be identified from reasoning on the world model.

3.2 Elements of the World Model

The world model consists of two parts. Figures 3 and 4 illustrate the world model for the diverter agent example.

The *situation model* (Fig. 3) holds knowledge about the agent situation. The situation of an agent consists both of its own characteristics and its relations to other entities in the world. The *domain models* (top) are models of the type of entities in the domain of the agent. They defines relevant classes of entities as well as relations between entities. For our example, we define that a diverter is connected to conveyors and can have a pallet located inside. Such concepts and relations can be extracted from existing models, such as the one presented in [12]. The *facts* (bottom) express the current knowledge about the world. Facts are expressed using the vocabulary defined by the models. They represent an abstraction of some meaningful aspects of the world, which can be used for

realizing high-level control tasks. For our example, facts express that D1 is a diverter, which is connected to conveyors C12, C25 and C33.

The *activity model* (Fig. 4) contains knowledge about the activities of the agent, i.e., the events and processes occurring in the world in which the agent is participating (as actor or observer). The *classification of activities*(top) models the types of activities in which the agent can be involved. Types are defined formally using description logic formulas and are organized hierarchically based on the subsumption relationship [2], noted $subClassOf$. Primitive types are defined as direct subclasses of Activity. Derived types are defined by restricting the primitive types to take into account the actual world of the agent. For instance, the generic type "Routing Pallet" is refined to more specific types like "Routing palletToDS2 on C25" corresponding to the case of the diverter agent.

The *expectations and observations* (bottom) model the activities that are expected and observed by the agent. Expectations and observations are defined by the specification of a type (based on the classification of activity types) and timing, expressed using time intervals [1]. Expectations are linked by dependencies, indicating how observations on one expectation can have consequences on other expectations. For instance, it is expected that "Routing palletToDS2 on C25", taking place between $t0$ and $t1$, requires both that "Observing Pallet entering from C12" takes place at $t0$ and "Observing Pallet leaving to C25" takes place at $t1$ (with a given tolerance). Additionally, it is expected that this activity would be prevented by "Detecting Switch Failure" during the same interval of time. Assuming that a pallet P123, with destination DS2, enters the diverter, an observation is added to the model, indicating that the activity "Observing P123 entering from C12" takes place at time 15.

3.3 Runtime Synchronization

The world model, and in particular the model of expectations and observations about activities, is synchronized incrementally, whenever new information is received from the LLC or from other agents. It is thus constantly evolving at runtime to reflect the current knowledge about the world as well as the current expectations that could be derived from this knowledge. Conversely, the changes in the model can be reflected in the underlying software, especially thanks to the LLC reconfiguration abilities.

Figure 5 depicts the general workflow for updating the model:

1. *Integration.* Whenever new information about the world is available, it is integrated in the world model by expressing the related type of activity and timing.
2. *Identification.* Forming a new observation requires identifying how it relates to existing expectations in the model. A matching is performed using the type and timing information. In case no expectation can be identified, an anomaly is reported.
3. *Propagation.* The addition of a new observation may trigger the creation of new expectations, Propagation occurs by considering dependencies be-

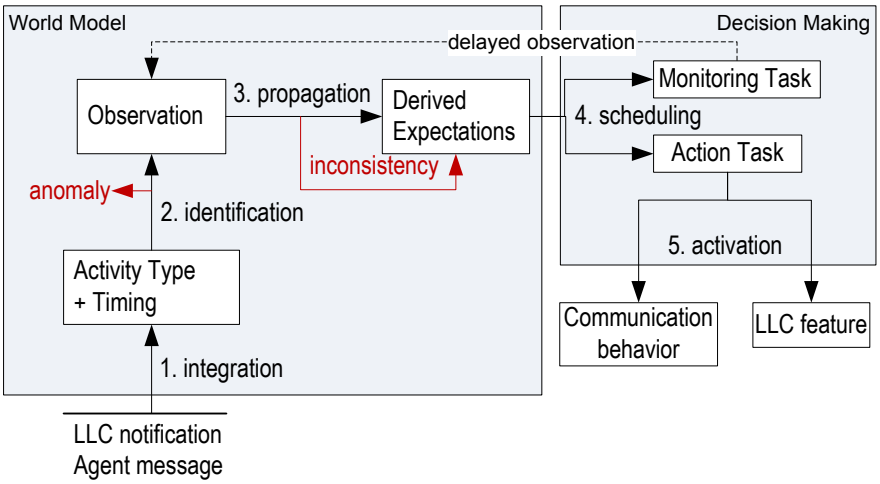


Fig. 5. Runtime synchronization of the world model

tween expectations and creating new expectations if needed. At this step, inconsistencies may be detected.

4. *Scheduling.* When new expectations are added to the world model, new decision-making tasks may be added to reflect them. We use monitoring tasks to trigger observations based on timing constraints (typically, these are observations about something that did not happen). Action tasks trigger external actions, either from the LLC or from other agents.
5. *Activation.* Relevant changes and anomalies in the activity model are notified to the decision making component, which is in charge of initiating the appropriate actions. Typical actions are setting up a communication behavior (i.e., initiating/terminating an interaction or cooperation protocol with other agents) or a LLC feature (i.e., adding/removing components in the LLC).

4 Results and Lessons Learned

We used the automation agent architecture and the world model as a basis for designing flexible manufacturing systems. In this section, we summarize our results by giving an overview of tasks involving the world model at runtime. We then discuss some important points and lessons learned.

4.1 Benefits of using a Model at Runtime

We designed the automation agent architecture and its world model to be generic and to apply to different classes of problems. Indeed, we could address several issues in a flexible manufacturing system using the model described in the previous section.

Detection of anomalies When interacting with the physical world, we are constantly faced with anomalies, disturbances and failures. Detecting anomalies before they cause more critical disturbances and failures is a definite advantage. Using its world model, an automation agent is able to detect anomalies which would otherwise be unnoticed by classical control software. For instance, we showed in [18] how an automation agent models its expectation about the completion of a transport task, and monitors relevant sensors to verify it. In case the sensor reading does not occur as expected, an anomaly is raised, indicating that a pallet is possibly stuck, or that the sensor in not working properly. Such a mechanism also benefits from the decentralized approach, enabling scalability and direct detection close to the relevant hardware.

Online diagnostics To enable the robust operation of a flexible manufacturing system, diagnostics and fault-recovery mechanisms are needed. The presented world model is especially helpful for the identifications of causes for a failures, and supports searching for explanations when an anomaly is observed [11]. This mechanism relies on defining expectations that could lead to the observation, and trying to verify them. Several directions can be exploited for verifying an expectation, for instance self-testing (e.g., trying to detect if a pallet was stuck by moving the switch to release it) and cooperation with other agents (e.g., asking a neighboring agent whether it detected a pallet which seems lost).

Runtime reconfiguration One of the most advanced features of a flexible manufacturing system is runtime reconfiguration, which is especially helpful to address challenges posed by an heterogeneous and continuously changing environment. As an example, in a pallet transport system, a destination may become unreachable due to the unexpected breakdown of a component. In order to keep the system running, we have studied solutions based on local reconfiguration, enabling conveyor belts to run in the opposite direction and intersections to modify their routing behavior. This requires a profound reconfiguration of the low-level control software, which our architecture allows. As presented in [9], the world model is directly involved in this task, both for identifying the need for reconfiguration and for preparing reconfiguration operations.

4.2 Lessons Learned

Besides the presented benefits, we can point out some lessons learned. They underline some important issues about the usage of models at runtime with networked physical devices, such as the ones encountered in a flexible manufacturing system. We identify three main challenges:

Challenge 1: Modeling dynamic aspects of the world. For dealing with a physical system, modeling dynamics is very important. A static view, even if regularly updated, is insufficient, as relevant information may not be accessible or integrated it in a timely manner. Modeling dynamic aspects enable

to obtain information from reasoning rather than from direct observation. Models and formal methods for managing time, time dependencies [21] as well as time imprecision are required.

Challenge 2: Synchronizing models incrementally. Physical devices operate under time constraints, and real-time execution is often incompatible with expensive model-based representation and reasoning. In order to cope with time constraints, we adopted an approach in which a fast LLC is fully responsible for performing all the functional operations of the system, while the slower HLC only performs complementary tasks to adapt and improve the behavior of the system. We found this approach suitable, but it also brings new challenges in terms of how the world model can reflect the reality while having only intermittent access to information from the world, and how it can synchronize to the real-world. Incremental synchronization of a model at runtime is essential [19].

Challenge 3: Integrating models in evolving systems. Working with networked physical devices requires the management of fragmented models over distributed agents. Moreover, large-scale systems require components to evolve independently. Ontologies are a general solution for interoperability [13], but are often unsuitable at runtime, since processing is overly complex. Considering that the system is rarely open, we consider just-in-time model-based generation of adapters and ad-hoc classifiers as more efficient, while ontologies provide a suitable abstraction for designers at design time.

5 Related Work

Model-driven engineering in gradually taking up in manufacturing systems [16]. However, these efforts mostly focus on models at design time, and do not seek to address issues regarding flexibility and robustness at runtime.

Previous works on using models at runtime are therefore highly relevant to our work. The general approach of using reasoning on a model to reconfigure a component-based system was already described by Oreizy *et al.* [14]. More recent effort have been focusing on modeling variability and adaptation in this approach in a generic way [5], providing a basis for the specification and validation of dynamic adaptive systems. One of the shortcomings for a direct application of such solutions in our domain is the lack of modeling of the dynamic behavior of the system, which we require for anticipating future states, detecting anomalies, as well as diagnosing past states in the presence of limited observations. Some works address more directly the behavioral modeling of some aspects of a dynamic adaptive system [20], as well as model-based runtime detection of errors [6]. However, we are not aware of a general solution for modeling activities among a distributed system of networked devices, which is required in our case.

6 Summary

In this paper, we presented an approach in which automation agents use models for efficiently monitoring and adapting the software controlling mechatronic

devices. Each agent manages a model integrating a representation of itself, of its environment and of the agent society, and enabling it to detect inconsistencies, to envision possible future states and to create explanations based on past states. We detailed the generic architecture and the model we use for representing the world of an agent. This model features a static part, called the situation model, and a dynamic part, called the activity model. One essential feature of our approach lies in the incremental synchronization of the activity model using information from low-level control software and from other agents.

As a further contribution of this paper, we presented results and lessons learned in this work. We have showed that the proposed approach using a model at runtime is the basis for monitoring and adapting control software in a flexible manufacturing systems. It provides significant improvement in terms of flexibility, robustness and performance. However, we point out that this approach raises new challenges regarding modeling dynamic aspects of the world, synchronizing models incrementally, and integrating models in evolving systems. Although our work partially addresses this challenges, further research in these directions is needed.

References

1. Allen, J.: Maintaining knowledge about temporal intervals. Communications of the ACM 26(11), 832–843 (1983)
2. Baader, F., Horrock, I., Sattler, U.: Description logics. In: Handbook on ontologies, pp. 3–28. Springer (2004)
3. Brennan, R.: Toward Real-Time distributed intelligent control: A survey of research themes and applications. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 37(5), 744–765 (2007)
4. Chituc, C.M., Restivo, F.J.: Challenges and trends in distributed manufacturing systems: Are wise engineering systems the ultimate answer? In: Second International Symposium on Engineering Systems MIT, Cambridge, Massachusetts (2009)
5. Fleurey, F., Dehlen, V., Bencomo, N., Morin, B., Jzquel, J.M.: Modeling and validating dynamic adaptation. In: Models in Software Engineering, pp. 97–108. Springer (2009), http://dx.doi.org/10.1007/978-3-642-01648-6_11
6. Hooman, J., Hendriks, T.: Model-based run-time error detection. In: Second International Workshop on Models@run.time (2007)
7. Jennings, N., Bussmann, S.: Agent-based control systems: Why are they suited to engineering complex systems? IEEE Control Systems Magazine 23(3), 61–73 (2003), http://dx.doi.org/10.1109/MCS.2003.1200249
8. Leitão, P.: Agent-based distributed manufacturing control: A state-of-the-art survey. Engineering Applications of Artificial Intelligence 22(7), 979–991 (2009), http://dx.doi.org/10.1016/j.engappai.2008.09.005
9. Lepuschitz, W., Zoitl, A., Vallée, M., Merdan, M.: Towards self-reconfiguration of manufacturing systems using automation agents. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews IN PRESS, 1–18 (2010), http://dx.doi.org/10.1109/TSMCC.2010.2059012
10. McFarlane, D., Marik, V., Valckenaers, P.: Guest editors' introduction: Intelligent control in the manufacturing supply chain. IEEE Intelligent Systems 20(1), 24–26 (2005), http://dx.doi.org/10.1109/MIS.2005.8

11. Merdan, M., Vallée, M., Lepuschitz, W., Zoitl, A.: Monitoring and diagnostics of industrial systems using automation agents. International Journal of Production Research Special Issue on Multi-agent and Holonic Techniques for Manufacturing Systems: Technologies and Applications, IN PRESS (2011)
12. Merdan, M., Koppensteiner, G., Hegny, I., Favre-Bulle, B.: Application of an ontology in a transport domain. In: IEEE International Conference on Industrial Technology (IEEE-ICIT 2008). Sichuan University, Chengdu, China (2008), http://dx.doi.org/10.1109/ICIT.2008.4608572
13. Obitko, M., Vrba, P., Mark, V., Radakovic, M.: Semantics in industrial distributed systems. In: IFAC (2006)
14. Oreizy, P., Gorlick, M.M., Taylor, R.N., Heimhigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D.S., Wolf, A.L.: An architecture-based approach to self-adaptive software. IEEE Intelligent Systems and Their Applications 14(3), 054–62 (1999), http://dx.doi.org/10.1109/5254.769885
15. Pěchouček, M., Mařík, V.: Industrial deployment of multi-agent technologies: review and selected case studies. Autonomous Agents and Multi-Agent Systems 17(3), 397–431 (Dec 2008), http://dx.doi.org/10.1007/s10458-008-9050-0
16. Strasser, T., Rooker, M., Hegny, I., Wenger, M., Zoitl, A., Ferrarini, L., Dede, A., Colla, M.: A research roadmap for model-driven design of embedded systems for automation components. In: 7th IEEE International Conference onIndustrial Informatics (INDIN 2009). pp. 564 –569 (jun 2009)
17. Vallée, M., Kaindl, H., Merdan, M., Lepuschitz, W., Arnautovic, E., Vrba, P.: An automation agent architecture with a reflective world model in manufacturing systems. In: IEEE International Conference on Systems, Man, and Cybernetics (SMC09). San Antonio, Texas, USA. (2009), http://dx.doi.org/10.1109/ICSMC.2009.5346161
18. Vallée, M., Merdan, M., Vrba, P.: Detection of anomalies in a transport system using automation agents with a reflective world model. In: Proceedings of the IEEE International Conference on Industrial Technologies (IEEE-ICIT 2010). pp. 489 – 494. Viña del Mar-Valparaso, Chile (2010), http://dx.doi.org/10.1109/ICIT.2010.5472751
19. Vogel, T., Neumann, S., Hildebrandt, S., Giese, H., Becker, B.: Incremental model synchronization for efficient run-time monitoring. In: Fourth International Workshop on Models@run.time (2009), http://ceur-ws.org/Vol-509/paper_8.pdf
20. Zhang, J., Cheng, B.H.C.: Model-based development of dynamically adaptive software. In: International Conference on Software Engineering (ICSE'06). China (2006), http://dx.doi.org/10.1145/1134285.1134337
21. Zhang, J., Cheng, B., , Goldsby, H.: Amoeba-rt: Run-time verification of adaptive software. In: Second International Workshop on Models@run.time (2007)
22. Zoitl, A.: Real-Time Execution for IEC 61499. No. ISBN: 978193439-4274, ISA-o3neidaA, USA (2009)